CHAPTER 3

LITERATURE REVIEW

3.1 INTRODUCTION

Open source software systems are becoming increasingly important these days. Many companies are investing in open source projects and lots of them are also using such software in their own work. Since the open source software developers do not employ any industrial standards, the quality and reliability of the code need to be studied. In the past, many researchers have delved into the process of analysing the open source system for fault finding and detecting its error proneness (Shen 1985). The inspiration for the work is drawn from the research works available on empirical analysis for validating the object-oriented software metrics.

3.2 REVIEW OF VALIDATING THE OBJECT-ORIENTED SOFTWARE METRICS

In the paper entitled, "An empirical validation of object oriented design metrics for fault predicton", Jie et al (2008) stated that they have validated object-oriented design metrics for defects estimation utilizing empirical analysis. The Chidamber and Kemerer metrics suite were utilized to assess the amount of defects in the programs. The approach involved statistical analysis and neuro-fuzzy techniques. The results pointed out that reliable defect estimation can be achieved by source lines of code , weighted methods per class, coupling between object classes and response for a class metrics. Particularly SLOC achieved the most prominent effect on the number of defects.

The authors Yuming and Hareton (2006) in “Empirical analysis of object oriented design metrics for predicting high and low severity faults” have taken fault severity into consideration utilizing the logistic regression and machine learning methods in their experimental exploration of the fault proneness predicting capability of object-oriented design metrics, particularly a subset of the Chidamber et al (1994) suite. The statistical relation across fault severity between most of these design metrics and fault-proneness of classes and the dependence of their prediction competence on severity of faults was revealed by the results obtained on a public domain National Aeronautics and Space Administration (NASA) data set. Also the results revealed that the fault-proneness prediction capabilities of these metrics vary considerably with the severity of the fault.

3.2.1 Object Oriented Function Points

In the paper titled “Object-oriented function points: An empirical validation”, Antoniol et al (2003) have experimentally verified size estimation models which are object-oriented. The experimental verification of Object Oriented Function Points (OOFP) has been extended considerably by utilizing a bigger data set and comparing OOFP with other predictors of LOC in their work. Various factors that affect size estimation were identified by a careful examination of the collected data points and developer practices, in addition to removing function point weighting tables from the OOFP process. Experimental results have proved that considerable enhancement in size estimates could be achieved by controlling these factors, 15% decrease of the normalized mean squared error corresponds to, a 56% reduction.

In “An empirical validation of object-oriented metrics in two different iterative software processes”- Mohammad and Wei (2003) stated that there were two iterative procedures for the experimental study of object oriented metrics. They are the short-cycled agile process and the long-cycled framework evolution process. Results showed that the design efforts and source lines of code added, changed, and deleted were successfully predicted by object oriented metrics in short-cycled agile process but the same features were not successfully predicted by it in the long-cycled framework process. This has proved that the design and implementation modifications during development iterations can be predicted by object oriented metrics whereas long-term development of an established system in diverse releases cannot be predicted by it.

In “empirical analysis of CK metrics for object-oriented design complexity: implications for software defects” - Ramanath Subramanyam and Krishnan (2003) provided with the experimental proof that a subset of the Chidamber and Kemerer suite, which are object oriented design complexity metrics, played an important role in identifying software defects. Experimental results on industry data belonging to software developed in two prevalent object oriented development programming languages showed that the metrics were considerably connected with defects even after controlling the software size. Moreover, effects of the metrics on defects varied over different samples from the two programming languages. Important inferences for designing high-quality object oriented software were provided by these results.

3.2.2 Three Object Oriented Metrics Suites

The paper titled “Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes” – authored by

Hector et al (2007) verified experimentally the software quality predicting capability regarding fault-proneness of three object oriented metrics suites. The three verified object oriented metrics suites were Chidamber and Kemerer metrics, Abreu's Metrics for object-oriented design (MOOD), and Bansiya & Davis' Quality Metrics for Object-Oriented Design (QMOOD). Defect data for six versions of Rhino, an open-source JavaScript application written in Java were used to determine the fault-prone classes predicting capability of the three metrics suites. The results proved that successful statistical models for detecting error-prone classes are produced by the CK and QMOOD suites which contain similar components. Good error-prone class predictors are not produced by the class components of the MOOD metrics suite.

DETECTING FAULT PRONENESS

Tibor et al (2005) in the paper, "Empirical validation of object-oriented metrics on open source software for fault prediction" - have demonstrated a procedure for detecting the fault-proneness of the source code of Mozilla which is an open source Web and e-mail suite, by illustrating the computing procedure of Chidamber and Kemerer object-oriented metrics. The usefulness of the metrics for fault-proneness prediction was verified by utilizing regression and machine learning methods to compare the obtained values with the amount of bugs present in its bug database known as Bugzilla. The variation in the predicted fault-proneness during the development cycle of the software system was identified by comparing the metrics of different versions of Mozilla.

In the research work titled, "An empirical study of system design instability metric and design evolution in an agile software process," - Mohammad and Wei li (2005) conducted an experimental study in two object-oriented (OO) systems, constructed utilizing an agile process that resembles Extreme Programming (XP) about the class growth and the System Design Instability (SDI) metrics. The daily collected evolutionary data of the two systems were analyzed. Their conclusion was that class growth of the systems pursued observable trends and project progress with some trends can be indicated by the SDI metric. They also concluded that correlation exists between SDI metric and XP activities. In early and late development phases, two consistent jumps in the SDI metric values were observed in both of the studied systems. Part of the results agreed with an earlier experimental study in a different environment.

3.3 MOTIVATION OF THE RESEARCH WORK

In “Leveraging open-source communities to improve the quality and performance of open-source software” – (Douglas 2000) stated that the Open-source development processes have emerged as an effective approach to reduce cycle-time and decrease design, implementation, and quality assurance costs for certain types of software, particularly systems infrastructure software, such as operating systems, compilers and language processing tools, editors, and distribution middleware. This paper presents two contributions to the study of open-source software engineering. First, they describe the key challenges of open-source software, such as controlling long-term maintenance and evolution costs, ensuring acceptable levels of quality, sustaining end-user confidence and good will, and ensuring the coherency of system-wide software and usability properties. They illustrated how well-organized open-source projects make it easier to address many of these challenges compared with traditional closed source approaches to building software. Second, they present the goals and methodology of the Skoll project, which focuses on developing and empirically validating novel open-source software quality assurance and optimization techniques to resolve key open-source challenges. They summarize the experimental design of a long-term case study of two widely used open-source middleware projects— Adaptive Communication Environment (ACE) and The Ace Orb (TAO)—that is used in the Skoll project to devise, deploy, and evaluate techniques for improving software quality through continuous distributed testing and profiling. These techniques are designed to leverage common open-source project assets, such as the technological sophistication and extensive computing resources of worldwide user communities, open access to source, and ubiquitous web access that can improve the quality and performance of open source software significantly.

In “Whither generic recovery from application faults? a fault study using open-source software” - Subhachandra and Peter (2000), tested the hypothesis that generic recovery techniques, such as process pairs, can survive most application faults without using application-specific information. They examined in detail the faults that occur in three, large, open-source applications: the Apache web server, the GNU/Linux and UNIX-type operating systems (GNOME) desktop environment, and the My SQL database. Using information contained in the bug reports and source code, they classified faults based on how they depend on the operating environment. They found that 72-87% of the faults are independent of the operating environment and are hence deterministic (non-transient). Recovering from the failures caused by these faults requires the use of application-specific knowledge. Half of the remaining faults depend on a condition in the operating environment that is likely to persist on retry, and the failures caused by these faults are also likely to require application-specific recovery. Unfortunately, only 5-14% of the faults were triggered by transient conditions, such as timing and synchronization that naturally fix them during recovery. That result indicates that classical application-generic recovery techniques, such as process pairs, were not sufficient to enable applications to survive most failures caused by application faults.

Nair et al (1998) described in his work, “A statistical assessment of some software testing strategies and application of experimental design techniques,” a case study of combination testing for a small subsystem of a screen-based administrative database. The system was designed to present users with input screens, accept data, then process it and store it in a database. Size was not given, but similar systems normally range from a few hundred to a few thousand lines of code. The study was extremely limited to one screen of a subsystem with two known faults involved, but pair wise testing was sufficient to detect both faults. In his works, “Failure modes in medical device software: an analysis of 15 years of recall data,” Wallace and Kuhn (2001) reviewed 15 years of medical device to recall data gathered by the US Food and Drug Administration (FDA) to characterize the types of faults that occur in the application domain. These applications include any devices under FDA authority, but are primarily small to medium sized embedded systems, and would range from roughly 104 to 105 lines of code.

All of the applications in the database were fielded systems that had been recalled because of reported defects. A limitation of the study, however, was that only 109 of the 342 recalls of software-controlled devices contained enough information to determine the number of conditions required to replicate a given failure. Of these 109 cases, 97 percent the reported flaws could be detected by testing all pairs of parameter settings and only three of the recalls had a Failure Triggering Fault Interaction (FTFI) number greater than 2. In “An Investigation of the applicability of design of experiments to software testing” the authors Kuhn and Reilly (2002) analyzed reports in bug tracking databases for open source browser and server software, the Mozilla web browser and Apache server. Both were early releases that were undergoing incremental development. The study found that more than 70 percent of documented failures were triggered by only one or two conditions, and that no failure had an FTFI number greater than 6. Difficulty in interpreting some of the failure reports led to conservative assumptions regarding failure causes. Thus, some of the failures with high FTFI numbers may actually have been less than 6.

The paper “Software fault interactions and implications for software testing” by Richard Kuhn (2004) proposed that Exhaustive testing of computer software is intractable, but empirical studies of software failures suggest that testing can in some cases be effectively exhaustive. Data reported in the study showed that software failures in a variety of domains were caused by combinations of relatively few conditions. These results have important implications for testing. If all faults in a system can be triggered by a combination of a fewer parameters, then testing all n-tuples of parameters is effectively equivalent to exhaustive testing, if software behavior is not dependent on complex event sequences and variables have a small set of discrete values.

3.3.1 Multiple Programming Interface

A large number of Multiple Programming Interface-like Multitasking (MPI) implementations are currently available, each of which emphasize different aspects of high-performance computing or are intended to solve a specific research problem. The result is a myriad of incompatible MPI implementations, all of which require separate installation, and the combination of which present significant logistical challenges for end users. Building upon prior research, and influenced by experience gained from the code bases of the Local Area Multicomputer MPI (LAM/MPI), Los Alamos (LA-MPI), and Fault Tolerant (FT-MPI) projects, Open MPI is an all-new production quality MPI-2 implementation that is fundamentally centered around component concepts. Open MPI provides a unique combination of novel features previously unavailable in an open-source, production-quality implementation of MPI. Its component architecture provides both a stable platform for third-party research as well as enabling the run-time composition of independent software add-ons. In a paper titled, “Open MPI: Goals, Concept and Design of a Next Generation MPI Implementation,” by Edgar et al (2004) presented a high-level overview of the goals, design, and implementation of Open MPI.

Many safety-critical systems rely on software to achieve their purposes. The number of such systems increases as additional capabilities are realized in software. Miniaturization and processing improvements have enabled the spread of safety-critical systems from nuclear and defense applications to domains as diverse as implantable medical devices, traffic control, smart vehicles, and interactive virtual environments. Future technological advances and consumer markets can be expected to produce more safety-critical applications. To meet this demand is a challenge. One of the major findings in a recent report by the President’s Information Technology Advisory Committee was, “The nation depends on fragile software”. Safety is a system problem. Software can contribute to a system’s safety or can compromise it by putting the system in a dangerous state. Software engineering of a safety-critical system thus requires a clear understanding of the software’s role in, and interactions with, the system (Robyn 2000).

3.3.2 Open Source Software Systems

The quality of the code needs to be measured and this can be done only with the help of proper tools. In this paper they have described a framework called Columbus with which they calculated the object oriented metrics validated by Basili et al (1996). For illustrating how fault-proneness detection from the open source web and e-mail suite called Mozilla can be done. They were also comparing the metrics of several versions of Mozilla to see how the predicted fault-proneness of the software system changed during its development. The Columbus framework has been further developed recently with a compiler wrapping technology that now gives us the possibility of automatically analyzing and extracting information from software systems without modifying any of the source codes or make files. This was also to introduce the fact of extraction process in “Software engineering for safety: A roadmap” – (Robyn 2000), to show what logic drives the various tools of the Columbus framework and what steps need to be taken to obtain the desired facts.

One area of the web services architecture yet to be standardized is that of fault tolerance for services. This was presented in the paper “Extracting facts from open source software” Rudolf et.al (2004). At the same time, Web Services Business Process Execution Language (WS-BPEL) is moving from a de facto standard to an Organization for the Advancement of Structured Information Standards (OASIS) ratified standard for combining services into processes. This paper investigates the feasibility of using WS-BPEL as an implementation technique for fault tolerant web services. The mapping of various fault tolerance patterns to WS-BPEL is presented. A prototype tool for combining service interfaces into a single façade and configuring fault tolerant mechanisms on a per-operation basis is also discussed. It is found that most fault tolerance patterns readily map onto WS-BPEL concepts, particularly when using the upcoming 2.0 version of the language. Evaluating and minimizing the performance overheads involved in process execution is identified as a key feature direction, as it works on the functionality and usability of the configuration tool.

Effective fault-management in the emerging complex distributed applications requires the ability to dynamically adapt resource allocation and fault tolerance policies in response to possible changes in environment, application requirements, and available resources. This paper presents an architecture framework of an Adaptive Fault Tolerance Management (AFTM) middleware using Common Object Request Broker Architecture (CORBA) compliant object request broker resting on the Solaris open system platform. In the paper presented by Glen Dobson (2003), “Using WS-BPEL to implement software fault tolerance for web services”. He discusses the approaches which have been tested through an AFTM prototyping effort.

Software errors are a major cause for system failures. To effectively design tools and support for detecting and recovering from software failures require a deep understanding of bug’s characteristics.

The study in “An approach fault tolerance in object-oriented open distributed systems” has discovered several new interesting characteristics: Memory-related bugs have decreased because quite a few effective detection tools became available recently; Surprisingly, some simple memory-related bugs such as NULL pointer dereferences that should have been detected by existing tools in development are still a major component, which indicates that the tools have not been used with their full capacity; Semantic bugs are the dominant root causes, as they are application specific and difficult to fix, which suggests that more efforts should be put into detecting and fixing them; Security bugs are increasing, and the majority of them cause severe impacts.

The paper named, “Have Things Changed now?” by Zhenmin et al (1998) proposed approaches to software testing based on methods from the field of design. These experiments have been advocated as a means of providing high coverage at relatively low cost. Tools to generate all pairs, or higher degree combinations, of input values have been developed and demonstrated in a few applications, but little empirical evidence is available to aid developers in evaluating the effectiveness of these tools for particular problems.

In “an investigation of the applicability of design of experiments to software testing,” D. Richard defined N-version programming as the independent generation of N2 2 functionally equivalent programs from the same initial specification (Richard 2006). A methodology of N-version programming has been devised and three types of special mechanisms have been identified that are needed to coordinate the execution of an N-version software unit and to compare the corresponding results generated by each version. Two experiments have been conducted to test the feasibility of N-version programming and the results of these experiments were discussed. In addition, constraints are identified and must be met for effective application of N-version programming.

Chen et al (1996) authored the paper titled, “N-Version Programming: A fault-tolerance approach to reliability of software operation”, where he describes the motivation, design and implementation of Berkeley Lab Checkpoint/Restart (BLCR), a system-level checkpoint/restart implementation for Linux clusters that targets the space of typical high performance computing applications, including MPI. Application-level solutions, including both check points and fault-tolerant algorithms, are recognized as more time and space efficient than system-level checkpoints, which cannot make use of any application-specific knowledge.

In “Failure analysis of open source J2EE application servers” by Junguo et al (2007), Open source J2EE application servers (J2EE ASs) have already attracted industrial attentions in recent years, but the reliability is still an obstacle to their wide acceptance. The detail of software failures in J2EE was seldom discussed in the past, although such information is valuable to evaluate and improve its reliability. In this paper, a measurement-based failure classification and analysis is presented. Presented results indicate open source J2EE AS's reliability needs improvement. Some notable results include: (1) the implementation of a widely used fault-tolerant mechanism, clustering, needs improvement; (2) only 15% of reported failures are removed within a week, and about 10% still open by the time I finish the study; (3) different J2EE ASs have different unreliable services so reliability improvement activities should be specific to both AS and applications.

In “An empirical study of open-source and closed-source software products” – Paulson et al (2004) describes the empirical study of open-source and closed-source software projects The motivation for this research is to quantitatively investigate common perceptions about open-source projects, and to validate these perceptions through an empirical study. This research work investigates the hypothesis that open-source software grows more quickly. The project growth is similar for all the projects in the analysis, indicating that other factors may limit growth. The hypothesis that creativity is more prevalent in open-source software is also examined, and evidence to support this hypothesis is found using the metric of functions added over time. The concept of open-source projects succeeding because of their simplicity is not supported by the analysis, nor is the hypothesis of open-source projects being more modular. However, the belief that defects are found and fixed more rapidly in open-source projects is supported by an analysis of the functions modified. This research work finds support for two of the five common beliefs and concludes that, when implementing or switching to the open-source development model, practitioners should ensure that an appropriate metrics collection strategy is in place to verify the perceived benefits.

In “Reliability issues in open source software” the authors Pandey and Vinay (2011) state that open source software in recent years has received great attention amongst software users. The success of the Linux Operating system, Apache web server and Mozilla web browser. demonstrates open source software development as an alternative form of software development. Despite the phenomenal success of open source software the reliability of OSS is often questioned. The opponents of open source software claim that open source software is unreliable as the source code of OSS is available and the potential threats can easily be incorporated. Whereas the supporters claim OSS to be more reliable than proprietary software as the source code is open and freely available for scrutiny for all. This paper analyzes the reliability issues of open source software in contrast to the proprietary software. Various views of researchers on the reliability of OSS are studied and analyzed and a theoretical study is made to examine the reliability of OSS. Results of various surveys on reliability conducted by various researchers/agencies are also incorporated in support of reliability analysis of OSS (Shooman 1972).

Arnaoudova et al (2010) in "Physical and conceptual identifier dispersion: measures and relation to fault proneness" state that poorly-chosen identifiers have been reported in the literature as misleading and increasing the program comprehension effort. Identifiers are composed of terms, which can be dictionary words, acronyms, contractions, or simple strings. The use of identical terms in different contexts may increase the risk of faults. This research work investigates the conjecture of using a measure combining the terms entropy and context coverage to study whether certain terms increase the odds ratios of methods to be fault-prone. Entropy measures the physical dispersion of terms in a program: the higher the entropy, the more scattered the terms are. Context coverage measures the conceptual dispersion of terms: the higher their context coverage, the more unrelated the methods using them. The proposed work computes the terms entropy and context coverage extracted from identifiers in Rhino 1.4R3 and ArgoUML 0.16. This research work shows statistically that methods containing terms with high entropy and context coverage are more fault-prone than others.

In “From Byzantine fault tolerance to intrusion tolerance” – Bessani et al (2011), demonstrated that a system is said intrusion-tolerant if it maintains its security properties despite some of its components being compromised by a malicious adversary. Although the implementation of these systems usually requires the use of Byzantine Fault-Tolerant (BFT) protocols, they are not a complete solution. Byzantine fault tolerance is a sub-field of fault tolerance research inspired by the Byzantine Generals' Problem. The object of Byzantine fault tolerance is to be able to defend against *Byzantine failures*, in which components of a system fail in arbitrary ways. Correctly functioning components of a Byzantine fault tolerant system will be able to correctly provide the system's service assuming there are not too many Byzantine faulty components. Besides BFT replication, there are several other techniques such as proactive recovery, diversity and confidential operation that are needed to implement these systems. Many of these techniques present interesting open problems that need to be addressed before a complete intrusion-tolerant system could be building and deployed.

In “Realization of user level fault tolerant management through a holistic approach for fault correlation”- Naughton and Agarwal (2011) state that many modern scientific applications, which are designed to utilize high performance parallel computers, occupy hundreds of thousands of computational cores running for days or even weeks. Since many scientists compete for resources, most supercomputing centers practice strict scheduling policies and perform meticulous accounting on their usage. Thus computing resources and time assigned to a user is considered invaluable. However, most applications are not well prepared for unforeseeable faults, still relying on primitive fault tolerance techniques. Considering that ever-plunging Mean Time to Interrupt (MTTI) is making scientific applications more vulnerable to faults, it is increasingly important to provide users not only an improved fault tolerant environment, but also a framework to support their own fault tolerance policies so that their allocation times can be best utilized. This paper addresses a user level fault tolerance policy management based on a holistic approach to digest and correlate fault related information. It introduces simple semantics with which users express their policies on faults, and illustrates how event correlation techniques can be applied to manage and determine the most preferable user policies. The paper also discusses an implementation of the framework using open source software, and demonstrates, as an example, how a molecular dynamics simulation application running on the institutional cluster at Oak Ridge National Laboratory benefits from it.

3.4 ORIGINALITY OF THE RESEARCH WORK

Object-oriented design has emerged as a dominant method in the software industry and many new metrics have been proposed for quality prediction of object-oriented programs, but the significance of those metrics is not yet confirmed. Software process control can be improved and high software reliability can be achieved if faults are predicted early in the software life cycle.

Testing the quality related issues of software has become critical with the increasing importance of the quality of software. Many authors have suggested theoretical validation followed by empirical evaluation using proven statistical and experimental techniques for evaluating in the field of usefulness and relevance of any new metrics (Victor 1995).

Also, to face the challenges in this software world, many of the companies release their developed software as open-source software. An open source software is a software that is released for downloading with free of cost. Some of the open source softwares are developed without any proper management. The quality and reliability of the software are not well designed.

An empirical validation of software quality metric suites on open source software for fault-proneness prediction in object oriented systems is presented in the research work. An efficient methodology to analysis and deduce the error in the open source package is proposed in this research work. After deducing the errors, error rate is to be calculated. Based on this calculated error rate, the package is detected to be useful or useless.

In some situation, if the package is detected to be useful, the path to handle the package is not known to the user. To overcome such difficulty, the way to deduce the class file and mapping of the JAR file is also proposed in this work. Open Source software JAVA is taken for the research work. Thus, this thesis proposes an efficient methodology for the use of open source software.

3.5 CONCLUSION AND REMARKS

Many reviews which are based on the research work were analyzed and presented. With the help of this analysis, the originality of the research work was proposed. In this research, the execution of the open source software is automated by picking up the suitable class path for execution. Through the proposed research work, Automated Execution of OSS (AEOSS) the client can execute the open source software in less time with less effort.

Based on this review of literature, the quality of the research work is improved, and thereby an efficient methodology to carry out the research is carried out.